8

Classes and Objects: A Deeper Look

Instead of this absurd division into sexes, they ought to class people as static and dynamic.
—Evelyn Waugh

Is it a world to hide virtues in?
—William Shakespeare

But what, to serve
our private ends,
Forbids the cheating
of our friends?
—Charles Churchill

This above all: to thine own self be true.
—William Shakespeare

Don't be "consistent," but be simply true.
—Oliver Wendell Holmes, Jr.

OBJECTIVES

In this chapter you will learn:

- Encapsulation and data hiding.
- The notions of data abstraction and abstract data types (ADTs).
- To use keyword `this`.
- To use `static` variables and methods.
- To import `static` members of a class.
- To use the `enum` type to create sets of constants with unique identifiers.
- How to declare `enum` constants with parameters.

Assignment Checklist

Name: ____________________ **Date:** ____________________

Section: ____________________

Exercises	Assigned: Circle assignments	Date Due
Prelab Activities		
Matching	YES NO	
Fill in the Blank	11, 12, 13, 14, 15, 16, 17, 18, 19, 20	
Short Answer	21, 22, 23, 24, 25, 26	
Programming Output	27, 28, 29, 30, 31, 32	
Correct the Code	33, 34, 35, 36, 37	
Lab Exercises		
Exercise 1 — Time: Part 1	YES NO	
Follow-Up Questions and Activities	1, 2	
Exercise 2 — Time: Part 2	YES NO	
Follow-Up Question and Activity	1	
Exercise 3 — Complex Numbers	YES NO	
Follow-Up Questions and Activities	1, 2	
Debugging	YES NO	
Labs Provided by Instructor		
1.		
2.		
3.		
Postlab Activities		
Coding Exercises	1, 2, 3, 4, 5, 6, 7	
Programming Challenges	1, 2	

Prelab Activities

Matching

Name: ______________________ **Date:** ______________________

Section: ______________________

After reading Chapter 8 of *Java How to Program: Sixth Edition*, answer the given questions. These questions are intended to test and reinforce your understanding of key Java concepts. You may answer these questions either before or during the lab.

For each term in the left column, write the letter for the description that best matches the term from the right column.

	Term	Description
___	1. composition	a) Such class members can be accessed by any class.
___	2. `enum` keyword	b) Method that is called by the garbage collector to clean up an object before it is removed from memory.
___	3. `public`	c) Such class members can be accessed only by the class in which they are defined.
___	4. `finalize`	d) “Has a” relationship.
___	5. mutator	e) Used implicitly in a class’s non-`static` methods to refer to both the instance variables and methods of an object of that class.
___	6. `private`	f) Together, this name and the class name compose the fully qualified name of the class.
___	7. attribute	g) Used to declare an enumeration class.
___	8. `static` import	h) Enables programmers to refer to `static` members as if they were declared in the class that uses them.
___	9. `this`	i) Another name for an instance variable in a class.
___	10. package name	j) Another name for a *set* method.

Prelab Activities

Name:

Fill in the Blank

Name: ____________________ **Date:** ____________________

Section: ____________________

Fill in the blanks for each of the following statements:

11. Keywords ________ and ________ are access modifiers.

12. Class members declared with access modifier ________ are accessible wherever the program has reference to an object of the class in which those members are defined.

13. Class members declared with access modifier ________ are accessible only to methods of the class in which those members are defined.

14. `enum` types are implicitly ________, because they declare constants that should not be modified. `enum` constants are implicitly ________.

15. There can be only one ________ declaration in each Java source-code file, and it must precede all other declarations and statements in the file.

16. A(n) ________ variable represents class-wide information.

17. In non-`static` methods, the keyword ________ is implicitly used to refer to the instance variables and other non-`static` methods of the class.

18. A(n) ________ initializes the instance variables of an object of a class when the object is instantiated.

19. Each class and interface in the Java API belongs to a specific ________ that contains a group of related classes and interfaces.

20. Instance variables are normally declared ________, and methods are normally declared ________.

Prelab Activities

Name:

Short Answer

Name: ____________________ **Date:** ____________________

Section: ____________________

Answer the following questions in the space provided. Your answers should be as concise as possible; aim for two or three sentences.

21. Why would a class provide overloaded constructors?

22. What are some advantages of creating packages?

23. What is the purpose of a constructor?

Prelab Activities

Name:

Short Answer

24. What is the purpose of a *set* method?

25. What is the purpose of a *get* method?

26. What is an abstract data type?

Prelab Activities

Name:

Programming Output

Name: __________________________ **Date:** __________________________

Section: __________________________

For each of the given program segments, read the code and write the output in the space provided below each program. [*Note:* Do not execute these programs on a computer.]

For questions 27–29 use the following declaration of class `Time2`:

```java
// Fig. 8.5: Time2.java
// Time2 class declaration with overloaded constructors.

public class Time2
{
   private int hour;   // 0 - 23
   private int minute; // 0 - 59
   private int second; // 0 - 59

   // Time2 no-argument constructor: initializes each instance variable
   // to zero; ensures that Time2 objects start in a consistent state
   public Time2()
   {
      this( 0, 0, 0 ); // invoke Time2 constructor with three arguments
   } // end Time2 no-argument constructor

   // Time2 constructor: hour supplied, minute and second defaulted to 0
   public Time2( int h )
   {
      this( h, 0, 0 ); // invoke Time2 constructor with three arguments
   } // end Time2 one-argument constructor

   // Time2 constructor: hour and minute supplied, second defaulted to 0
   public Time2( int h, int m )
   {
      this( h, m, 0 ); // invoke Time2 constructor with three arguments
   } // end Time2 two-argument constructor

   // Time2 constructor: hour, minute and second supplied
   public Time2( int h, int m, int s )
   {
      setTime( h, m, s ); // invoke setTime to validate time
   } // end Time2 three-argument constructor

   // Time2 constructor: another Time2 object supplied
   public Time2( Time2 time )
   {
      // invoke Time2 three-argument constructor
      this( time.getHour(), time.getMinute(), time.getSecond() );
   } // end Time2 constructor with a Time2 object argument

```

Fig. L 8.1 | `Time2.java` (Part 1 of 3.)

Prelab Activities

Name:

Programming Output

```
   // Set Methods
   // set a new time value using universal time; ensure that
   // the data remains consistent by setting invalid values to zero
   public void setTime( int h, int m, int s )
   {
      setHour( h );   // set the hour
      setMinute( m ); // set the minute
      setSecond( s ); // set the second
   } // end method setTime

   // validate and set hour
   public void setHour( int h )
   {
      hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
   } // end method setHour

   // validate and set minute
   public void setMinute( int m )
   {
      minute = ( ( m >= 0 && m < 60 ) ? m : 0 );
   } // end method setMinute

   // validate and set second
   public void setSecond( int s )
   {
      second = ( ( s >= 0 && s < 60 ) ? s : 0 );
   } // end method setSecond

   // Get Methods
   // get hour value
   public int getHour()
   {
      return hour;
   } // end method getHour

   // get minute value
   public int getMinute()
   {
      return minute;
   } // end method getMinute

   // get second value
   public int getSecond()
   {
      return second;
   } // end method getSecond

   // convert to String in universal-time format (HH:MM:SS)
   public String toUniversalString()
   {
      return String.format(
         "%02d:%02d:%02d", getHour(), getMinute(), getSecond() );
   } // end method toUniversalString

   // convert to String in standard-time format (H:MM:SS AM or PM)
   public String toString()
   {
```

Fig. L 8.1 | `Time2.java` (Part 2 of 3.)

Prelab Activities

Name:

Programming Output

```java
      return String.format( "%d:%02d:%02d %s",
         ( (getHour() == 0 || getHour() == 12) ? 12 : getHour() % 12 ),
         getMinute(), getSecond(), ( getHour() < 12 ? "AM" : "PM" ) );
   } // end method toString
} // end class Time2
```

Fig. L 8.1 | `Time2.java` (Part 3 of 3.)

27. What is output by the following code segment?

```java
Time3 t1 = new Time3( 5 );
System.out.printf( "The time is %s\n", t1 );
```

Your answer:

28. What is output by the following code segment?

```java
Time3 t1 = new Time3( 13, 59, 60 );
System.out.printf( "The time is %s\n", t1 );
```

Your answer:

Prelab Activities

Name:

Programming Output

29. What is output by the following code segment?

```
Time3 t1 = new Time3( 0, 30, 0 );
Time3 t2 = new Time3( t1 );
System.out.printf( "The time is %s\n", t2.toUniversalString() );
```

Your answer:

For questions 30–32 use the following declaration of class Person:

```
public class Person
{
   private String firstName;
   private String lastName;
   private String gender;
   private int age;

   public Person( String firstName, String lastName )
   {
      setName( firstName, lastName );
      setGender( "n/a" );
      setAge( -1 );
   } // end Person constructor

   public Person( String firstName, String lastName, String gender, int age )
   {
      setName( firstName, lastName );
      setGender( gender );
      setAge( age );
   } // end Person constructor

   public void setName( String firstName, String lastName )
   {
      this.firstName = firstName;
      this.lastName = lastName;
   } // end method setName

   public void setGender( String gender )
   {
      this.gender = gender;
   } // end method setGender

```

Fig. L 8.2 | (Part 1 of 2.)

Prelab Activities

Name:

Programming Output

```
   public void setAge( int age )
   {
      this.age = age;
   } // end method setAge

   public String getName()
   {
      return String.format( “%s %s”, firstName, lastName );
   } // end method getName

   public String getGender()
   {
      return gender;
   } // end method getGender

   public int getAge()
   {
      return age;
   } // end method getAge

   public String toString()
   {
      if ( gender == "n/a" && age == -1 )
         return getName();

      return String.format( “%s is a %d year old %s”, getName(), getAge(),
         getGender() );
   } // end method toString
}  // end class Person
```

Fig. L 8.2 | (Part 2 of 2.)

30. What is output by the following code segment?

```
Person person = new Person( "Rus", "Tic", "male", 21 );
System.out.println( person );
```

Your answer:

Prelab Activities

Name:

Programming Output

31. What is output by the following code segment?

```
Person person = new Person( "Anna Lee", "Tic" );
System.out.println( person );
```

Your answer:

32. What is output by the following code segment?

```
Person person = new Person( "Anna Lee", "Tic", "n/a", -1 );
System.out.println( person );
```

Your answer:

Correct the Code

Name: ____________________ **Date:** ____________________

Section: ____________________

Determine if there is an error in each of the following program segments. If there is an error, specify whether it is a logic error or a compilation error, circle the error in the program and write the corrected code in the space provided after each problem. If the code does not contain an error, write "no error." [*Note*: There may be more than one error in each program segment.]

33. The following defines class `Product`, with a no-argument constructor that sets the product's name to an empty `String` and the `price` to 0.00, and a `toProductString` method that returns a `String` containing the product's `name` and its `price`:

```
public class Product
{
   private String name;
   private double price;

   public void Product()
   {
      name = "";
      price = 0.00;
   } // end Product constructor

   public toString()
   {
      return String.format( “%s costs %.2d”, name, price );
   } // end method toString
} // end class Product
```

Your answer:

Prelab Activities

Name:

Correct the Code

34. The following defines another constructor for class `Product` that takes two arguments and assigns those arguments to the corresponding instance variables:

```
public Product( String name, double price )
{
   name = name;
   price = price;
}
```

Your answer:

35. The following defines two *set* methods to set the `name` and the `price` of the `Product`:

```
public setName()
{
   this.name = name;
}

public setPrice()
{
   this.price = price;
}
```

Your answer:

Prelab Activities

Name:

Correct the Code

36. The following code segment should create a Product object and display a String containing the values of the object’s instance variables.

```
Product p1 = new Product( "Milk", 5.5 );
System.out.printf( “%s %.2f\n”, p1.name, p1.price );
```

Your answer:

37. The following code segment should create a Product object, set the values of its instance variables and display a String containing the values of the instance variables:

```
Product p1 = new Product();
p1.setName();
p1.setPrice();
System.out.println( p1.toString( "Eggs", 3 ) );
```

Your answer:

Lab Exercises

Lab Exercise 1 — Time: Part 1

Name: ____________________________ **Date:** ____________________________

Section: ____________________________

The following problem is intended to be solved in a closed-lab session with a teaching assistant or instructor present. The problem is divided into six parts:

1. Lab Objectives
2. Problem Description
3. Sample Output
4. Program Template (Fig. L 8.3–Fig. L 8.4)
5. Problem-Solving Tips
6. Follow-Up Questions and Activities

The program template represents a complete working Java program with one or more key lines of code replaced with comments. Read the problem description and examine the output, then study the template code. Using the problem-solving tips as a guide, replace the `/* */` comments with Java code. Compile and execute the program. Compare your output with the sample output provided. Then answer the follow-up questions. The source code for the template is available at `www.deitel.com` and `www.prenhall.com/deitel`.

Lab Objectives

This lab was designed to reinforce programming concepts from Chapter 8 of *Java How To Program: Sixth Edition*. In this lab, you will practice:

- Modifying methods of a class.
- Accessing instance variables.
- Using *set* and *get* methods.

The follow-up questions and activities also will give you practice:

- Understanding the difference between access specifiers `public` and `private`.

Problem Description

Modify the *set* methods in class `Time2` of Fig. L 8.1 to return appropriate error values if an attempt is made to set one of the instance variables `hour`, `minute` or `second` of an object of class `Time` to an invalid value. [*Hint:* Use `boolean` return types on each method.] Write a program that tests these new *set* methods and outputs error messages when incorrect values are supplied.

Lab Exercises

Name:

Lab Exercise 1 — Time: Part 1

Sample Output

```
1. Set Hour
2. Set Minute
3. Set Second
4. Add 1 second
5. Exit
Choice: 1
Enter Hours: 10
Hour: 10  Minute: 0  Second: 0
Universal time: 10:00:00   Standard time: 10:00:00 AM
1. Set Hour
2. Set Minute
3. Set Second
4. Add 1 second
5. Exit
Choice: 2
Enter Minutes: 10
Hour: 10  Minute: 10  Second: 0
Universal time: 10:10:00   Standard time: 10:10:00 AM
1. Set Hour
2. Set Minute
3. Set Second
4. Add 1 second
5. Exit
Choice: 3
Enter Seconds: 10
Hour: 10  Minute: 10  Second: 10
Universal time: 10:10:10   Standard time: 10:10:10 AM
1. Set Hour
2. Set Minute
3. Set Second
4. Add 1 second
5. Exit
Choice: 3
Enter Seconds: 99
Invalid seconds.
Hour: 10  Minute: 10  Second: 0
Universal time: 10:10:00   Standard time: 10:10:00 AM
1. Set Hour
2. Set Minute
3. Set Second
4. Add 1 second
5. Exit
Choice: 5
```

Lab Exercises

Name:

Lab Exercise 1 — Time: Part 1

Template

```
// Lab 1: Time2.java
// Time2 class definition with methods tick,
// incrementMinute and incrementHour.

public class Time2
{
   private int hour; // 0 - 23
   private int minute; // 0 - 59
   private int second; // 0 - 59

   // Time2 no-argument constructor: initializes each instance variable
   // to zero; ensures that Time2 objects start in a consistent state
   public Time2()
   {
      this( 0, 0, 0 ); // invoke Time2 constructor with three arguments
   } // end Time2 no-argument constructor

   // Time2 constructor: hour supplied, minute and second defaulted to 0
   public Time2( int h )
   {
      this( h, 0, 0 ); // invoke Time2 constructor with three arguments
   } // end Time2 one-argument constructor

   // Time2 constructor: hour and minute supplied, second defaulted to 0
   public Time2( int h, int m )
   {
      this( h, m, 0 ); // invoke Time2 constructor with three arguments
   } // end Time2 two-argument constructor

   // Time2 constructor: hour, minute and second supplied
   public Time2( int h, int m, int s )
   {
      setTime( h, m, s ); // invoke setTime to validate time
   } // end Time2 three-argument constructor

   // Time2 constructor: another Time2 object supplied
   public Time2( Time2 time )
   {
      // invoke Time2 constructor with three arguments
      this( time.getHour(), time.getMinute(), time.getSecond() );
   } // end Time2 constructor with Time2 argument

   // Set a new time value using universal time. Perform
   // validity checks on data. Set invalid values to zero.
   /* Write header for setTime. */
   {
      /* Write code here that declares three boolean variables which are
         initialized to the return values of setHour, setMinute and setSecond.
         These lines of code should also set the three member variables. */

      /* Return true if all three variables are true; otherwise, return false. */
   }

```

Fig. L 8.3 | `Time2.java`. (Part 1 of 3.)

Lab Exercises

Name:

Lab Exercise 1 — Time: Part 1

```
   // validate and set hour
   /* Write header for the setHour method. */
   {
      /* Write code here that determines whether the hour is valid.
         If so, set the hour and return true. */

      /* If the hour is not valid, set the hour to 0 and return false. */
   }

   // validate and set minute
   /* Write the header for the setMinute method. */
   {
      /* Write code here that determines whether the minute is valid.
         If so, set the minute and return true. */

      /* If the minute is not valid, set the minute to 0 and return false. */
   }

   // validate and set second
   /* Write the header for the setSecond method. */
   {
      /* Write code here that determines whether the second is valid.
         If so, set the second and return true. */

      /* If the second is not valid, set the second to 0 and return false. */
   }

   // Get Methods
   // get hour value
   public int getHour()
   {
      return hour;
   } // end method getHour

   // get minute value
   public int getMinute()
   {
      return minute;
   } // end method getMinute

   // get second value
   public int getSecond()
   {
      return second;
   } // end method getSecond

   // Tick the time by one second
   public void tick()
   {
      setSecond( second + 1 );

      if ( second == 0 )
         incrementMinute();
   } // end method tick

```

Fig. L 8.3 | `Time2.java`. (Part 2 of 3.)

Lab Exercises

Name:

Lab Exercise 1 — Time: Part 1

```
   // Increment the minute
   public void incrementMinute()
   {
      setMinute( minute + 1 );

      if ( minute == 0 )
         incrementHour();
   } // end method incrementMinute

   // Increment the hour
   public void incrementHour()
   {
      setHour( hour + 1 );
   } // end method incrementHour

   // convert to String in universal-time format (HH:MM:SS)
   public String toUniversalString()
   {
      return String.format(
         "%02d:%02d:%02d", getHour(), getMinute(), getSecond() );
   } // end method toUniversalString

   // convert to String in standard-time format (H:MM:SS AM or PM)
   public String toString()
   {
      return String.format( "%d:%02d:%02d %s",
         ( ( getHour() == 0 || getHour() == 12 ) ? 12 : getHour() % 12 ),
         getMinute(), getSecond(), ( getHour() < 12 ? "AM" : "PM" ) );
   } // end method toStandardString
} // end class Time2
```

Fig. L 8.3 | Time2.java. (Part 3 of 3.)

```
// Lab 1: Time2Test.java
// Program adds validation to Fig. 8.7 example
import java.util.Scanner;

public class Time2Test
{
   public static void main( String args[] )
   {
      Scanner input = new Scanner( System.in );

      Time2 time = new Time2(); // the Time2 object

      int choice = getMenuChoice();

      while ( choice != 5 )
      {
         switch ( choice )
         {
            case 1: // set hour
               System.out.print( "Enter Hours: " );
               int hours = input.nextInt();

```

Fig. L 8.4 | Time2Test.java. (Part 1 of 2.)

Lab Exercises

Name:

Lab Exercise 1 — Time: Part 1

```
               /* Write code here that sets the hour. If the hour is invalid,
                  display an error message. */

               break;
            case 2: // set minute
               System.out.print( "Enter Minutes: " );
               int minutes = input.nextInt();

               /* Write code here that sets the minute. If the minute is invalid,
                  display an error message. */

               break;
            case 3: // set seconds
               System.out.print( "Enter Seconds: " );
               int seconds = input.nextInt();

               /* Write code here that sets the second. If the second is invalid,
                  display an error message. */

               break;
            case 4: // add 1 second
               time.tick();
               break;
         } // end switch

         System.out.printf( "Hour: %d  Minute: %d  Second: %d\n",
            time.getHour(), time.getMinute(), time.getSecond() );
         System.out.printf( "Universal time: %s   Standard time: %s\n",
            time.toUniversalString(), time.toString() );

         choice = getMenuChoice();
      } // end while
   } // end main

   // prints a menu and returns a value corresponding to the menu choice
   private static int getMenuChoice()
   {
      Scanner input = new Scanner( System.in );

      System.out.println( "1. Set Hour" );
      System.out.println( "2. Set Minute" );
      System.out.println( "3. Set Second" );
      System.out.println( "4. Add 1 second" );
      System.out.println( "5. Exit" );
      System.out.print( "Choice: " );

      return input.nextInt();
   } // end method getMenuChoice
} // end class Time2Test
```

Fig. L 8.4 | `Time2Test.java`. (Part 2 of 2.)

Problem-Solving Tips

1. Use `boolean` return types for the *set* methods.
2. Each *set* method should `return true` if the value is valid and `false` if it is not.
3. If you have any questions as you proceed, ask your lab instructor for assistance.

Lab Exercises

Name:

Lab Exercise 1 — Time: Part 1

Follow-Up Questions and Activities

1. What is the purpose of declaring the instance variables of class `Time2` `private`?
2. Change all the methods in class `Time2` from `public` methods to `private` methods, then try to recompile the class and execute the program again. Does anything occur differently? If so, explain why.

Lab Exercises

Lab Exercise 2 — Time: Part 2

Name: ____________________________ **Date:** ____________________________

Section: ____________________________

The following problem is intended to be solved in a closed-lab session with a teaching assistant or instructor present. The problem is divided into six parts:

1. Lab Objectives
2. Problem Description
3. Sample Output
4. Program Template (Fig. L 8.5–Fig. L 8.6)
5. Problem-Solving Tips
6. Follow-Up Question and Activity

The program template represents a complete working Java program with one or more key lines of code replaced with comments. Read the problem description and examine the output, then study the template code. Using the problem-solving tips as a guide, replace the `/* */` comments with Java code. Compile and execute the program. Compare your output with the sample output provided. Then answer the follow-up question. The source code for the template is available at `www.deitel.com` and `www.prenhall.com/deitel`.

Lab Objectives

This lab was designed to reinforce programming concepts from Chapter 8 of *Java How To Program: Sixth Edition*. In this lab you will practice:

- Creating new methods in a class.
- Calling methods of a class from the class's other methods.

The follow-up question and activity also will give you practice:

- Understanding modularization.

Problem Description

Modify class `Time2` of Fig. L 8.1 to include a `tick` method that increments the time stored in a `Time2` object by one second. Provide method `incrementMinute` to increment the minute and method `incrementHour` to increment the hour. The `Time2` object should always remain in a consistent state. Write a program that tests the `tick` method, the `incrementMinute` method and the `incrementHour` method to ensure that they work correctly. Be sure to test the following cases:

a) incrementing into the next minute,

b) incrementing into the next hour and

c) incrementing into the next day (i.e., 11:59:59 PM to 12:00:00 AM).

Lab Exercises

Name:

Lab Exercise 2 — Time: Part 2

Sample Output

```
Enter the time
Hours: 23
Minutes: 59
Seconds: 59
1. Add 1 second
2. Add 1 Minute
3. Add 1 Hour
4. Add seconds
5. Exit
Choice: 1
Hour: 0  Minute: 0  Second: 0
Universal time: 00:00:00   Standard time: 12:00:00 AM
1. Add 1 second
2. Add 1 Minute
3. Add 1 Hour
4. Add seconds
5. Exit
Choice: 2
Hour: 0  Minute: 1  Second: 0
Universal time: 00:01:00   Standard time: 12:01:00 AM
1. Add 1 second
2. Add 1 Minute
3. Add 1 Hour
4. Add seconds
5. Exit
Choice: 3
Hour: 1  Minute: 1  Second: 0
Universal time: 01:01:00   Standard time: 1:01:00 AM
1. Add 1 second
2. Add 1 Minute
3. Add 1 Hour
4. Add seconds
5. Exit
Choice: 4
Enter seconds to tick: 60
Hour: 1  Minute: 2  Second: 0
Universal time: 01:02:00   Standard time: 1:02:00 AM
1. Add 1 second
2. Add 1 Minute
3. Add 1 Hour
4. Add seconds
5. Exit
Choice: 5
```

Lab Exercises

Name:

Lab Exercise 2 — Time: Part 2

Template

```java
// Lab 2: Time2.java
// Time2 class definition with methods tick,
// incrementMinute and incrementHour.

public class Time2
{
   private int hour; // 0 - 23
   private int minute; // 0 - 59
   private int second; // 0 - 59

   // Time2 no-argument constructor: initializes each instance variable
   // to zero; ensures that Time2 objects start in a consistent state
   public Time2()
   {
      this( 0, 0, 0 ); // invoke Time2 constructor with three arguments
   } // end Time2 no-argument constructor

   // Time2 constructor: hour supplied, minute and second defaulted to 0
   public Time2( int h )
   {
      this( h, 0, 0 ); // invoke Time2 constructor with three arguments
   } // end Time2 one-argument constructor

   // Time2 constructor: hour and minute supplied, second defaulted to 0
   public Time2( int h, int m )
   {
      this( h, m, 0 ); // invoke Time2 constructor with three arguments
   } // end Time2 two-argument constructor

   // Time2 constructor: hour, minute and second supplied
   public Time2( int h, int m, int s )
   {
      setTime( h, m, s ); // invoke setTime to validate time
   } // end Time2 three-argument constructor

   // Time2 constructor: another Time2 object supplied
   public Time2( Time2 time )
   {
      // invoke Time2 constructor with three arguments
      this( time.getHour(), time.getMinute(), time.getSecond() );
   } // end Time2 constructor with Time2 argument

   // Set Methods
   // set a new time value using universal time; perform
   // validity checks on data; set invalid values to zero
   public void setTime( int h, int m, int s )
   {
      setHour( h ); // set the hour
      setMinute( m ); // set the minute
      setSecond( s ); // set the second
   } // end method setTime

```

Fig. L 8.5 | `Time2.java`. (Part 1 of 3.)

Lab Exercises

Name:

Lab Exercise 2 — Time: Part 2

```java
   // validate and set hour
   public void setHour( int h )
   {
      hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
   } // end method setHour

   // validate and set minute
   public void setMinute( int m )
   {
      minute = ( ( m >= 0 && m < 60 ) ? m : 0 );
   } // end method setMinute

   // validate and set second
   public void setSecond( int s )
   {
      second = ( ( s >= 0 && s < 60 ) ? s : 0 );
   } // end method setSecond

   // Get Methods
   // get hour value
   public int getHour()
   {
      return hour;
   } // end method getHour

   // get minute value
   public int getMinute()
   {
      return minute;
   } // end method getMinute

   // get second value
   public int getSecond()
   {
      return second;
   } // end method getSecond

   // Tick the time by one second
   /* Write header for method tick */
   {
      /* Write code that increments the second by one, then determines whether
         the minute needs to be incremented. If so, call incrementMinute. */
   }

   // Increment the minute
   /* Write header for method incrementMinute */
   {
      /* Write code that increments the minute by one, then determines whether
         the hour needs to be incremented. If so, call incrementHour. */
   }

   // Increment the hour
   /* Write header for method incrementHour. */
   {
      /* Write code that increments the hour by one. */
   }
```

Fig. L 8.5 | Time2.java. (Part 2 of 3.)

Lab Exercises

Name:

Lab Exercise 2 — Time: Part 2

```java

   // convert to String in universal-time format (HH:MM:SS)
   public String toUniversalString()
   {
      return String.format(
         "%02d:%02d:%02d", getHour(), getMinute(), getSecond() );
   } // end method toUniversalString

   // convert to String in standard-time format (H:MM:SS AM or PM)
   public String toString()
   {
      return String.format( "%d:%02d:%02d %s",
         ( ( getHour() == 0 || getHour() == 12 ) ? 12 : getHour() % 12 ),
         getMinute(), getSecond(), ( getHour() < 12 ? "AM" : "PM" ) );
   } // end method toStandardString
} // end class Time2
```

Fig. L 8.5 | `Time2.java`. (Part 3 of 3.)

```java
// Lab 2: Time2Test.java
// Demonstrating the Time2 class set and get methods
import java.util.Scanner;

public class Time2Test
{
   public static void main( String args[] )
   {
      Scanner input = new Scanner( System.in );

      Time2 time = new Time2();

      System.out.println( "Enter the time" );
      System.out.print( "Hours: " );
      time.setHour( input.nextInt() );
      System.out.print( "Minutes: " );
      time.setMinute( input.nextInt() );
      System.out.print( "Seconds: " );
      time.setSecond( input.nextInt() );

      int choice = getMenuChoice();

      while ( choice != 5 )
      {
         switch ( choice )
         {
            case 1: // add 1 second
               time.tick();
               break;
            case 2: // add 1 minute
               time.incrementMinute();
               break;
            case 3: // and 1 hour
               time.incrementHour();
               break;
```

Fig. L 8.6 | `Time2Test.java`. (Part 1 of 2.)

Lab Exercises

Name:

Lab Exercise 2 — Time: Part 2

```
            case 4: // add arbitrary seconds
               System.out.print( "Enter seconds to tick: " );
               int ticks = input.nextInt();

               for ( int i = 0; i < ticks; i++ )
                  time.tick();

               break;
         } // end switch

         System.out.printf( "Hour: %d  Minute: %d  Second: %d\n",
            time.getHour(), time.getMinute(), time.getSecond() );
         System.out.printf( "Universal time: %s   Standard time: %s\n",
            time.toUniversalString(), time.toString() );

         choice = getMenuChoice();
      } // end while
   } // end main

   // prints a menu and returns a value corresponding to the menu choice
   private static int getMenuChoice()
   {
      Scanner input = new Scanner( System.in );

      System.out.println( "1. Add 1 second" );
      System.out.println( "2. Add 1 Minute" );
      System.out.println( "3. Add 1 Hour" );
      System.out.println( "4. Add seconds" );
      System.out.println( "5. Exit" );
      System.out.print( "Choice: " );

      return input.nextInt();
   } // end method getMenuChoice
} // end class Time2Test
```

Fig. L 8.6 | `Time2Test.java`. (Part 2 of 2.)

Problem-Solving Tips

1. Use the *set* methods of class `Time2` to assign new values to the appropriate `Time2` instance variables.
2. The `tick` and increment methods do not return anything; therefore, they should be declared to return `void`.
3. Complete your testing by running the application and testing all three cases mentioned in the problem description. Note that methods `incrementMinute` and `incrementHour` can be tested by changing the time to a value for which the next call to tick will cause either (or both) of these methods to be called. For example, at 11:59:59, the next tick will cause both the hour and minute to be incremented.
4. If you have any questions as you proceed, ask your lab instructor for assistance.

Follow-Up Question and Activity

1. Explain why a programmer would choose to implement method `tick` in class `Time2` rather than a class that uses `Time2` objects.

Lab Exercises

Name:

Lab Exercise 3 — Complex Numbers

Name: ______________________ **Date:** ______________________

Section: ______________________

The following problem is intended to be solved in a closed-lab session with a teaching assistant or instructor present. The problem is divided into six parts:

1. Lab Objectives
2. Problem Description
3. Sample Output
4. Program Template (Fig. L 8.7–Fig. L 8.8)
5. Problem-Solving Tips
6. Follow-Up Questions and Activities

The program template represents a complete working Java program with one or more key lines of code replaced with comments. Read the problem description and examine the output, then study the template code. Using the problem-solving tips as a guide, replace the `/* */` comments with Java code. Compile and execute the program. Compare your output with the sample output provided. Then answer the follow-up questions. The source code for the template is available at `www.deitel.com` and `www.prenhall.com/deitel`.

Lab Objectives

This lab was designed to reinforce programming concepts from Chapter 8 of *Java How To Program: Sixth Edition*. In this lab, you will practice:

- Using the `this` reference.
- Initializing class objects.
- Using overloaded constructors.

The follow-up questions and activities will also give you practice:

- Enhancing a class's functionality with a new method.

Problem Description

Create a class called `Complex` for performing arithmetic with complex numbers. Complex numbers have the form

$$realPart + imaginaryPart * i$$

where i is

$$\sqrt{-1}$$

Write a program to test your class. Use floating-point variables to represent the `private` data of the class. Provide a constructor that enables an object of this class to be initialized when it is declared. Provide a no-argument constructor with default values in case no initializers are provided. Provide `public` methods that perform the following operations:

a) Add two `Complex` numbers: The real parts are added together and the imaginary parts are added together.

b) Subtract two `Complex` numbers: The real part of the right operand is subtracted from the real part of the left operand, and the imaginary part of the right operand is subtracted from the imaginary part of the left operand.

c) Print `Complex` numbers in the form `(a, b)`, where `a` is the real part and `b` is the imaginary part.

Lab Exercises

Name:

Lab Exercise 3 — Complex Numbers

Sample Output

```
a = (9.5, 7.7)
b = (1.2, 3.1)
a + b = (10.7, 10.8)
a - b = (8.3, 4.6)
```

Template

```java
// Lab 3: Complex.java
// Definition of class Complex

public class Complex
{
   private double real;
   private double imaginary;

   // Initialize both parts to 0
   /* Write header for a no-argument constructor. */
   {
      /* Write code here that calls the Complex constructor that takes 2
         arguments and initializes both parts to 0 */
   } // end Complex no-argument constructor

   // Initialize real part to r and imaginary part to i
   /* Write header for constructor that takes two arguments--real part r and
      imaginary part i. */
   {
      /* Write line of code that sets real part to r. */
      /* Write line of code that sets imaginary part to i. */
   }

   // Add two Complex numbers
   public Complex add( Complex right )
   {
      /* Write code here that returns a Complex number in which the real part is
         the sum of the real part of this Complex object and the real part of the
         Complex object passed to the method; and the imaginary part is the sum
         of the imaginary part of this Complex object and the imaginary part of
         the Complex object passed to the method. */
   }

   // Subtract two Complex numbers
   public Complex subtract( Complex right )
   {
      /* Write code here that returns a Complex number in which the real part is
         the difference between the real part of this Complex object and the real
         part of the Complex object passed to the method; and the imaginary part
         is the difference between the imaginary part of this Complex object and
         the imaginary part of the Complex object passed to the method. */
   }
```

Fig. L 8.7 | `Complex.java`. (Part 1 of 2.)

Lab Exercises

Name:

Lab Exercise 3 — Complex Numbers

```

   // Return String representation of a Complex number
   public String toString()
   {
      return String.format( "(%.1f, %.1f)", real, imaginary );
   } // end method toComplexString;
} // end class Complex
```

Fig. L 8.7 | `Complex.java`. (Part 2 of 2.)

```
// Exercise 8.12: ComplexTest.java
// Test the Complex number class

public class ComplexTest
{
   public static void main( String args[] )
   {
      // initialize two numbers
      Complex a = new Complex( 9.5, 7.7 );
      Complex b = new Complex( 1.2, 3.1 );

      System.out.printf( "a = %s\n", a );
      System.out.printf( "b = %s\n", b );
      System.out.printf( "a + b = %s\n", a.add( b ) );
      System.out.printf( "a - b = %s\n", a.subtract( b ) );
   } // end main
} // end class ComplexTest
```

Fig. L 8.8 | `ComplexTest.java`

Problem-Solving Tips

1. For the `add` and `subtract` methods of class `Complex`, return a new `Complex` object with the results of the calculations.
2. If you have any questions as you proceed, ask your lab instructor for assistance.

Follow-Up Questions and Activities

1. In the `ComplexTest` class of *Lab Exercise 3*, instead of adding `b` to `a`, add `a` to `b`. Also instead of subtracting `b` from `a`, subtract `a` from `b`. Are the results different from the previous results in *Lab Exercise 3*?
2. In class `Complex`, define a `multiply` method that returns the product of two `Complex` numbers. Suppose you are trying to compute the product of two complex numbers $a + bi$ and $c + di$. The real part will be $ac - bd$, while the imaginary part will be $ad + bc$. Modify `ComplexTest` to test your solution.

Lab Exercises

Name:

Debugging

Name: _____________________________ Date: _____________________________

Section: _____________________________

The program in this section does not compile. Fix all the compilation errors so that the program will compile successfully. Once the program compiles, execute the program, and compare the output with the sample output. Then eliminate any logic errors that may exist. The sample output demonstrates what the program's output should be once the code is corrected. The source code is available at `www.deitel.com` and at `www.prenhall.com/deitel`.

Sample Output

```
Monthly balances for one year at .04
Balances:
             Saver 1   Saver 2
Base        $2000.00  $3000.00
Month 1:    $2006.67  $3010.00
Month 2:    $2013.36  $3020.03
Month 3:    $2020.07  $3030.10
Month 4:    $2026.80  $3040.20
Month 5:    $2033.56  $3050.33
Month 6:    $2040.33  $3060.50
Month 7:    $2047.14  $3070.70
Month 8:    $2053.96  $3080.94
Month 9:    $2060.81  $3091.21
Month 10:   $2067.68  $3101.51
Month 11:   $2074.57  $3111.85
Month 12:   $2081.48  $3122.22

After setting interest rate to .05
Balances:
Saver 1      Saver 2
$2090.16    $3135.23
```

Broken Code

```java
// Exercise 8.6 solution: SavingAccount
// SavingAccount class definition

public class SavingAccount
{
   // interest rate for all accounts
   private static double annualInterestRate = 0;

   private final double savingsBalance; // balance for currrent account

   // constructor, creates a new account with the specified balance
   public void SavingAccount( double savingsBalance )
   {
      savingsBalance = savingsBalance;
   } // end constructor

```

Lab Exercises

Name:

Debugging

```
   // get monthly interest
   public void calculateMonthlyInterest()
   {
      savingsBalance += savingsBalance * ( annualInterestRate / 12.0 );
   } // end method calculateMonthlyInterest

   // modify interest rate
   public static void modifyInterestRate( double newRate )
   {
      annualInterestRate =
         ( newRate >= 0 && newRate <= 1.0 ) ? newRate : 0.04;
   } // end method modifyInterestRate

   // get string representation of SavingAccount
   public String toString()
   {
      return String.format( "$%.2f", savingsBalance );
   } // end method toSavingAccountString
} // end class SavingAccount
```

```
// Exercise 8.6 solution: SavingAccountTest.java
// Program that tests SavingAccount class

public class SavingAccountTest
{
   public static void main( String args[] )
   {
      SavingAccount saver1 = new SavingAccount( 2000 );
      SavingAccount saver2 = new SavingAccount( 3000 );
      SavingAccount.annualInterestRate = 0.04;

      System.out.println( "Monthly balances for one year at .04" );
      System.out.println( "Balances:" );

      System.out.printf( "%20s%10s\n", "Saver 1", "Saver 2" );
      System.out.printf( "%-10s%10s%10s\n", "Base",
         saver1.toString(), saver2.toString() );

      for ( int month = 1; month <= 12; month++ )
      {
         String monthLabel = String.format( "Month %d:", month );
         saver1.calculateMonthlyInterest();
         saver2.calculateMonthlyInterest();

         System.out.printf( "%-10s%10s%10s\n", monthLabel,
            saver1.toString(), saver2.toString() );
      } // end for

      SavingAccount.modifyInterestRate( .05 );
      saver1.calculateMonthlyInterest();
      saver2.calculateMonthlyInterest();

      System.out.println( "\nAfter setting interest rate to .05" );
      System.out.println( "Balances:" );
      System.out.printf( "%-10s%10s\n", "Saver 1", "Saver 2" );
```

Lab Exercises

Name:

Debugging

```
      System.out.printf( "%-10s%10s\n",
         saver1.toString(),  saver2.toString() );
   } // end main
} // end class SavingAccountTest
```

Postlab Activities

Coding Exercises

Name: ____________________________ **Date:** ____________________________

Section: ____________________________

These coding exercises reinforce the lessons learned in the lab and provide additional programming experience outside the classroom and laboratory environment. They serve as a review after you have successfully completed the *Prelab Activities* and *Lab Exercises*.

For each of the following problems, write a program or a program segment that performs the specified action:

1. Write the class declaration for class `Square` that has a `private` instance variable `side` of type `double` and a no-argument constructor that sets the `side` to `1.0` by calling a method named `setSide` that you will declare in *Coding Exercise 2*.

2. Write a method `setSide` for the class you defined in *Coding Exercise 1*. Set the `side` variable to the argument of the method. Also make sure that the `side` is not less than `0.0`. If it is, keep the default setting of `1.0`.

Postlab Activities

Name:

Coding Exercises

3. Write a method `getSide` for the class you modified in *Coding Exercise 2* that retrieves the value of instance variable `side`.

4. Define another constructor for the class that you modified in *Coding Exercise 3* that takes one argument, the `side`, and uses the `Square`'s *set* method to set the `side`.

5. Write a method `computeArea` for the class that you modified in *Coding Exercise 4* that computes the area of a `Square`.

Postlab Activities

Name:

Coding Exercises

6. Define a `toString` method for the class that you modified in *Coding Exercise 5* that will return a `String` containing the value of `side` and the area of the `Square`.

7. Define application class `SquareTest` to test the `Square` class you defined in *Coding Exercises 1–6*. Ensure that all your methods and constructors work properly.

Postlab Activities

Name:

Programming Challenges

Name: ____________________ **Date:** ____________________

Section: ____________________

The *Programming Challenges* are more involved than the *Coding Exercises* and may require a significant amount of time to complete. Write a Java program for each of the problems in this section. The answers to these problems are available at `www.deitel.com` and `www.prenhall.com/deitel`. Pseudocode, hints or sample outputs are provided for each problem to aid you in your programming.

1. Create a class `Rectangle`. The class has attributes `length` and `width`, each of which defaults to 1. Provide methods that calculate the `perimeter` and the `area` of the rectangle. Provide *set* and *get* methods for both `length` and `width`. The *set* methods should verify that `length` and `width` are each floating-point numbers greater than or equal to `0.0` and less than `20.0`. Write a program to test class `Rectangle`.

Hints:

- This class is very similar to the class you developed in the *Coding Exercises* section.
- Your output should appear as follows:

```
1. Set Length
2. Set Width
3. Exit
Choice: 1
Enter length: 10
Length: 10.00
Width: 1.00
Perimeter: 22.00
Area: 10.00
1. Set Length
2. Set Width
3. Exit
Choice: 2
Enter width: 15
Length: 10.00
Width: 15.00
Perimeter: 50.00
Area: 150.00
1. Set Length
2. Set Width
3. Exit
Choice: 1
Enter length: 99
Length: 1.00
Width: 15.00
Perimeter: 32.00
Area: 15.00
1. Set Length
2. Set Width
3. Exit
Choice: 3
```

Postlab Activities

Name:

Programming Challenges

2. Create a more sophisticated `Rectangle` class than the one you created in *Programming Challenge 1*. This class stores only the Cartesian coordinates of the four corners of the rectangle. The constructor calls a *set* method that accepts four sets of coordinates and verifies that each of these is in the first quadrant with no single *x*- or *y*-coordinate larger than `20.0`. The *set* method also verifies that the supplied coordinates specify a rectangle. Provide methods to calculate the `length`, `width`, `perimeter` and `area`. The length is the larger of the two dimensions. Include a predicate method `isSquare` which determines whether the rectangle is a square. Write a program to test class `Rectangle`.

Hint:

- Your output should appear as follows:

```
Enter rectangle's coordinates
x1: 10
y1: 8
x2: 10
y2: 1
x3: 1
y3: 1
x4: 1
y4: 8
Length: 9.00
Width: 7.00
Perimeter: 32.00
Area: 63.00
```